foeXlab-Materialien

Arduino-Schaltungen

First Steps

Gottfried Wilhelm Leibniz Universität Hannover

Inhaltsverzeichnis

1 Das Board 3

2 Programmierung: Grundstruktur 3

3 Versuche 4

3.1 Eine LED blinkt 4
3.2 Eine LED leuchtet – jetzt mit Schalter 4
3.3 Mit Schalter und LEDs zur Ampel 5
3.4 Ein Abstandssensor wird hinzugefügt 5

Impressum 7

Bildverzeichnis 7

1 Das Board

Ein Mikrocontroller (Atmel ATmega1280) ist der Kern des Arduino Mega. Zusammen mit einigen Zusatzbeschaltungen für die Eingänge und Ausgänge, einem Schwingquarz und der Stromversorgung bildet der Arduino einen programmierbaren Minicomputer. Die Programmiersprache C oder C++ ist geeignet, diesen zu programmieren. Der Arduino kann, bei entsprechender Programmierung aber auch mit einem PC in Verbindung treten. Der Arduino stellt alle notwendigen Softwaretools, wie z. B. einen Editor kostenfrei zur Verfügung.

Die Programmier-Verbindung des Arduino nach draußen ist seriell, findet also bitweise statt. Wir verwenden die inzwischen gebräuchliche USB-Schnittstelle. Die Umwandlung USB nach seriell übernimmt ein zusätzlicher Wandler-Chip. Auf dem Board befinden sich Steckerleisten, mit den für die Verbindungen zuständigen I/O-Pins: 54 digitale Pins und 16 analoge. „PWM“ steht für Pulsweitenmodulation, an diesen 15 Pins ist eine Regelung der Spannung in Schritten möglich. Pins ohne PWM besitzen die beiden Zustände „high“ (Spannung ein, 5 V) oder „low“ (Spannung aus, 0 V).

2 Programmierung: Grundstruktur

Die Programmierung erfolgt in der Arduino eigenen Software. C/C++ als Programmiersprache ist case-sensitiv, unterscheidet also zwischen Groß- und Kleinbuchstaben.

(1) Definition eventueller Variablen, vor Beginn der Funktionen
(2) setup( )-Funktion (wird genau einmal durchlaufen)
(3) loop( )-Funktion; dieser Code wird in einer nicht abbrechenden Schleife abgearbeitet; hier werden beispielsweise die Eingabe- und Ausgabebefehle abgearbeitet.

Beispiel setup( ) und loop( ):

```
void setup ()
{
        pinMode (pin, OUTPUT);  //Hier wird ‚pin‘ als Ausgang definiert
}
```

```
void loop ( )
{
        digitalWrite (pin, HIGH);   //schaltet ‚pin‘ ein (= high)
        delay (1000);                       // 1000 ms = 1 s Pause
        digitalWrite (pin, LOW);    // schaltet ‚pin‘ aus (= low)
        delay (1000);                       // 1000 ms Pause
}
```

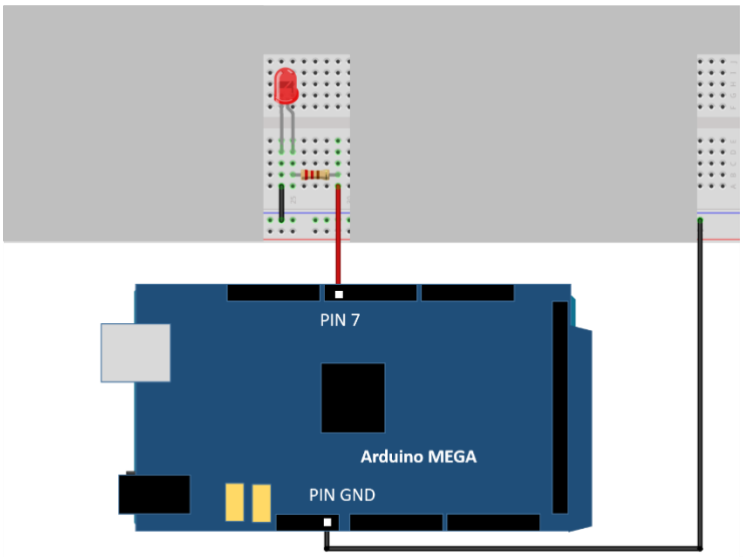
Die geschweiften Klammern { und } rahmen Blöcke mit Anweisungen ein, Anfang und Ende. Das Semikolon trennt die Anweisungen voneinander. Zeilenwechsel dienen nur der Übersichtlichkeit: Einzeilige Kommentare gehen bis zum Zeilenende und beginnen mit //. Ganze Kommentarblöcke, die vom Programm ignoriert werden sollen, fangen mit /* an und hören mit */ auf.
Ist ein Programm fertig geschrieben, wird mit einem Klick auf das hinterlegte Häkchen der Programmcode auf Fehler überprüft und das Programm erstellt. Mit dem danebenliegenden Rechtspfeil wird es auf den Arduino übertragen.

3 Versuche

3.1 Eine LED blinkt

Als erster Versuch soll eine LED im Sekundenabstand blinken. Die Schaltung hierzu wird nach dem folgenden Schema aufgebaut:

Um einzelne Pins anzusteuern, lässt sich die Nummer dieser verwenden. Sobald das Programm etwas umfangreicher wird, ist dies jedoch ein sehr fehleranfälliges Verfahren. Hierzu werden Variablen u.a. verwendet, als Stellvertreternamen für die Pin-Nummern. Je nachdem wie groß die zu speichernden Daten sind, wird der Variablentyp gewählt – für diesen Fall reicht der Typ „int“ (Integer).

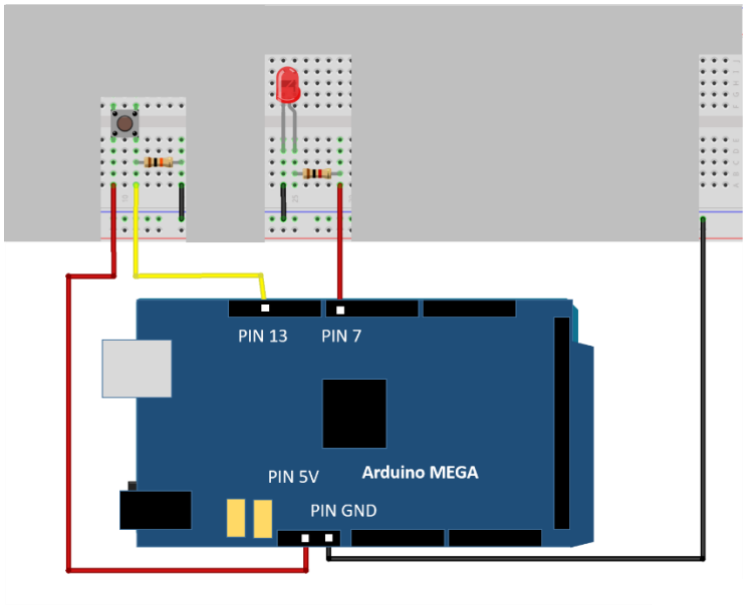



```
int       ledRot = 7;                  //Pin 7 wird ledRot genannt

void setup ( )
{
          pinMode (ledRot, OUTPUT);    //Pin 7 wird als Ausgang definiert
}

void loop ( )
{
          digitalWrite (ledRot, HIGH);  //ledRot wird auf high gesetzt: die LED also eingeschaltet
          delay (1000);                 // 1000 ms = 1 s Pause
          digitalWrite (ledRot, LOW);   //ledRot wird auf low gesetzt: die LED also ausgeschaltet
          delay (1000);                 // 1 s Pause
}
```

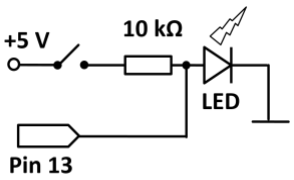
3.2 Eine LED leuchtet – jetzt mit Schalter

Um mehr Einfluss auf die LED zu haben, wird nun ein Schalter nach dem folgenden Schema hinzugefügt (als Widerstand hierfür bitte den mit „Taster“ markierten verwenden):

Den Schalter verwenden wir im folgenden Programm als Eingangssignal. Im Normalzustand ist der Taster auf 0 V (der sogenannte „Pulldown“-Widerstand zieht das Signal in diesem Fall auf 0 V herunter). Das Einschalten der LED erfolgt erst beim drücken des Schalters. Um das drücken zu erkennen, wird in der loop-Funktion permanent überprüft ob am entsprechenden Pin eine Spannung angelegt (= der Schalter gedrückt) wird. Dies erfolgt über eine Wenn… dann... (if/else) Abfrage. Das doppelte Gleichheitszeichen „==“ dient hierbei als Vergleichsoperator.




```
int     ledRot = 7;                     //Pin 7 wird ledRot genannt
int     schalter = 13;                  //Pin 13 soll Eingangspin werden
int     schalterstatus=0;           //verwendet um drücken des Schalters zu erkennen

void setup()
{
        pinMode(ledRot, OUTPUT);
        pinMode(schalter, INPUT);
}

void loop()
{
        schalterstatus=digitalRead(schalter); //Auslesen des Wertes von Pin 13, speichern in Variable
        if (schalterstatus == HIGH)             //wenn Schalter gedrückt, dann...
        {
                digitalWrite(ledRot, HIGH);
                delay (5000);
                digitalWrite(ledRot, LOW);
        }
        else                                    //ansonsten...
        {
                digitalWrite(ledRot, LOW);      //LED weiterhin ausgeschaltet
        }
}
```



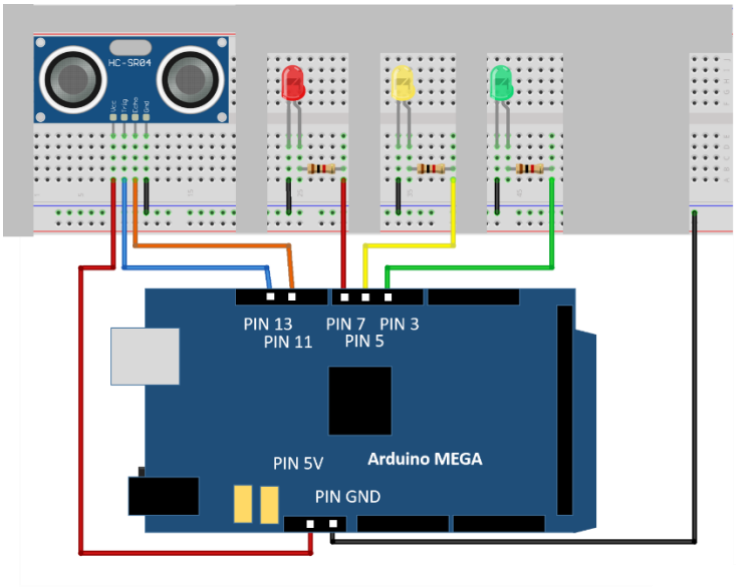
3.3 Mit Schalter und LEDs zur Ampel

Mit den bisher verwendeten Befehlen und Schaltungen sollte es nun auch ohne explizite Anleitung möglich sein eine Ampelschaltung in den typischen Farben aufzubauen. Alles Notwendige hierzu findet sich in den beiden vorherigen Versuchen.

3.4 Ein Abstandssensor wird hinzugefügt

Um den Schalter nicht mehr direkt drücken zu müssen, wird ein Ultraschall-Abstandssensor an dessen Stelle nach dem folgenden Schema eingebaut (Ampelschaltung nur beispielhaft eingefügt):

Die Anschlüsse Vcc (+) und Gnd (-) dienen hierbei der Stromversorgung des Sensors, Trig (für Trigger) sendet ein Signal über den einen Lautsprecher aus und Echo misst die Laufzeit bis die (reflektierte) Schallwelle wieder beim zweiten Lautsprecher angekommen ist. Hierbei ist zu beachten, dass die Laufzeit noch durch zwei geteilt werden muss, da die Welle den Weg doppelt zurücklegt (Hin- und Rückweg). Der Faktor 0.03432 dient der Umrechnung der Laufzeit mit Schallgeschwindigkeit in eine Entfernung in Zentimeter.



Das folgende Programm enthält nur die zusätzlich zum bisherigen Code notwendigen Bestandteile.

```
int       trigger=13;
int       echo=11;
long      laufzeit=0;              /*long wird verwendet, da die Messwerte größer sein können
                                   als die Speicherkapazität des int-Typs */
long      entfernung=0;

void setup()
{
          pinMode(trigger, OUTPUT);
          pinMode(echo, INPUT);
          ...
}

void loop()
{
          digitalWrite(trigger, LOW); //Spannung vom Trigger entfernen, damit rauschfreies Signal
          delay(5);                   //5 ms reichen hierbei
          digitalWrite(trigger, HIGH);          //Senden des Ultraschallsignals
          delay(10);                            //für 10 ms
          digitalWrite(trigger, LOW);
          laufzeit = pulseIn(echo, HIGH); /*mit pulseIn wird die Zeit gemessen, bis Signal am zweiten
                                            Lautsprecher wieder eintrifft und „laufzeit“ zugeordnet*/
          entfernung = (laufzeit/2) * 0.03432; //Berechnung der Entfernung in cm
          if (entfernung <= 10)                 //ist die Entfernung kleiner oder gleich (zB) 10 cm, dann...
          {
                    …                           //auslösen der Ampel-Schaltung wie vorab beim Schalter
          }
}
```

Impressum

Arduino-Schaltungen
First Steps

herausgegeben von
foeXlab

bearbeitet von
Malte Below

Bildverzeichnis

alle Abbildungen Archiv DRS